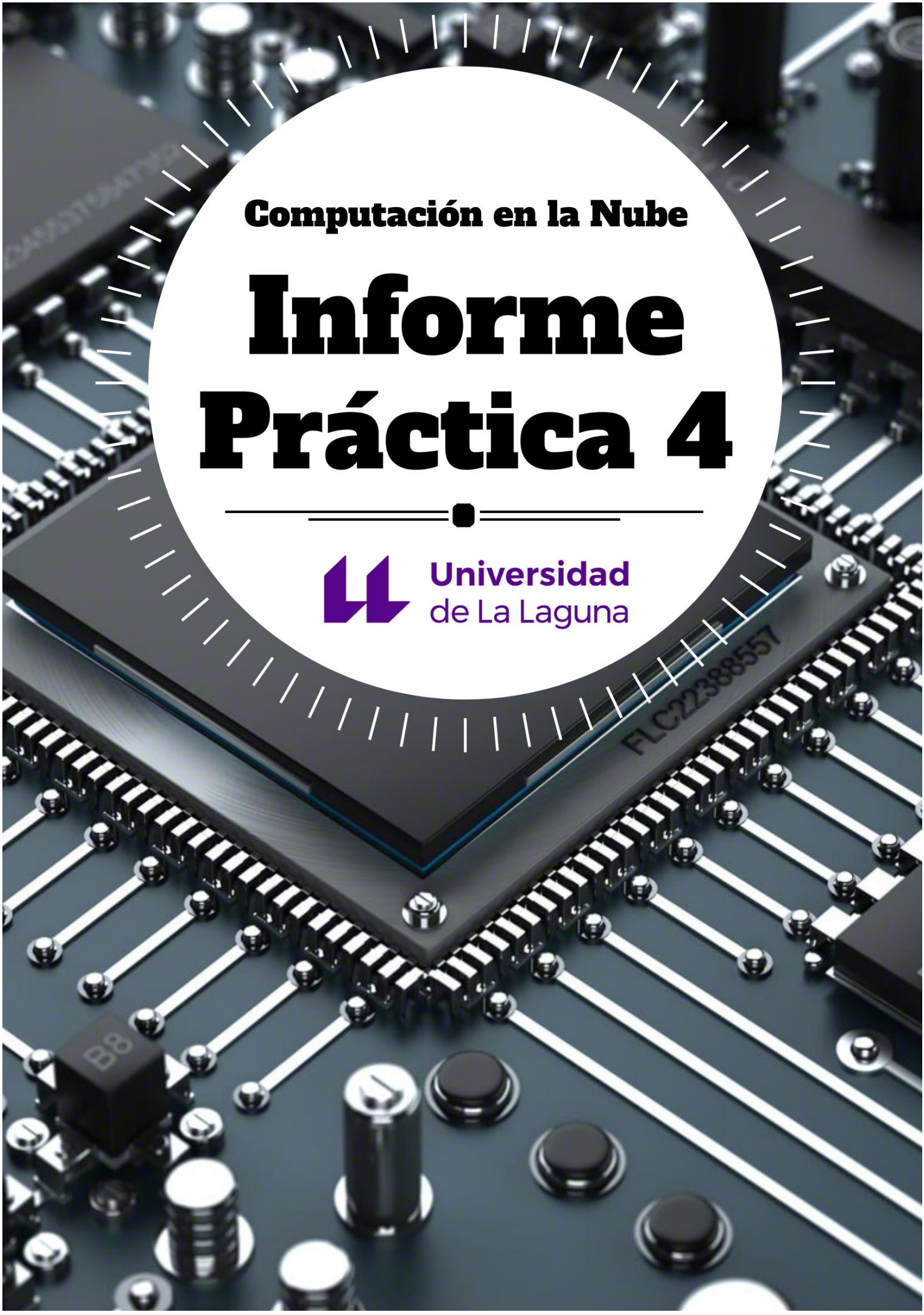


Computación en la Nube

Informe Práctica 4

Universidad de La Laguna

Práctica 4

1. Desarrolla una versión en CUDA del código que has desarrollado en la práctica 3 para el procesamiento de imágenes.

Tras un intento fallido con Cuda. Se ha procedido a usar OpenCL de forma satisfactoria. Solamente una función se ha modificado con respecto a la versión anterior. Esta función, es la que vamos a proporcionarle a la gráfica para que realice el filtrado correspondiente. Para poder realizar esto, se ha añadido a un nuevo fichero denominado “clProgram.cl”. De igual forma, se ha cambiado el planteamiento de dicha función para que aplique el filtrado a un vector en específico. Esta función la podremos ver a continuación:

```
#pragma OPENCL EXTENSION cl_khr_fp64 : enable

kernel void applyFilter(global double *image, global double *filter,
                        global double *newImage, int height, int width,
                        int filterHeight, int filterWidth)
{
    int newImageHeight = height - filterHeight + 1;
    int newImageWidth = width - filterWidth + 1;
    int d, j, h, w;

    int i = get_global_id(0);

    for ( d = 0; d < 3; d++)
    {
        for (j = 0; j < newImageWidth; j++)
        {
            float sum = 0, val = 0;
            for (h = i; h < i + filterHeight; h++)
            {
                for (w = j; w < j + filterWidth; w++)
                {
                    float weight = filter[ (h - i) * filterWidth + (w - j)];
                    val += image[d * height * width + h * width + w] * weight;
                    sum += weight;
                }

                newImage[d * newImageHeight * newImageWidth + i * newImageWidth + j] = round(val / sum);
            }
        }
    }
```

Posteriormente, se ha modificado el “main” del código principal adaptándolo a la sección de OpenCl. Para ello, se han creado las siguientes modificaciones:

Inicialización de OpenCL

```
std::vector<cl::Platform> all_platforms;
cl::Platform::get(&all_platforms);

if ( all_platforms.size() == 0 ) {
    printf("No se han encontrado plataformas para ejecutar OpenCL");
    exit(1);
}

cl::Platform default_platform = all_platforms[0];
std::cout << "Using platform: "<< default_platform.getInfo<CL_PLATFORM_NAME>() << "\n";

std::vector<cl::Device> all_devices;

default_platform.getDevices(CL_DEVICE_TYPE_ALL, &all_devices);

if (all_devices.size() == 0 ) {
    printf("La plataforma escogida no es compatible con OpenCL");
    exit(1);
}

cl::Device default_device = all_devices[0];
std::cout << "Using device: " << default_device.getInfo<CL_DEVICE_NAME>() << "\n";

cl::Context context({default_device});

cl::Program::Sources sources;

int size = 0;
char *kernel_code = readCLSource("clProgram.cl", size);

sources.push_back({kernel_code, size});

cl::Program program(context, sources);

if( program.build({default_device})!=CL_SUCCESS ) {
    std::cout <<" Error building: " << program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(default_device) << "\n";
    exit(1);
}
```

Preparación de datos en OpenCL

```
vector<double> sourceCPU;
vector<double> resultCPU;
vector<double> filtroCPU;

for (int d = 0; d < 3; d++)
{
    for (int i = 0; i < image[0].size(); i++)
    {
        for (int j = 0; j < image[0][0].size(); j++)
        {
            sourceCPU.push_back(image[d][i][j]);
        }
    }
}

for ( int i = 0; i < filterHeight; i++) {
    for ( int j = 0; j < filterWidth; j++) {
        filtroCPU.push_back(filter[i][j]);
    }
}

resultCPU.resize(3 * newImageHeight * newImageWidth);

// BUFFERS DE OPENCL
cl::Buffer sourceGPU(context, CL_MEM_READ_WRITE, sourceCPU.size() * sizeof(double));
cl::Buffer resultGPU(context, CL_MEM_READ_WRITE, resultCPU.size() * sizeof(double));
cl::Buffer filtroGPU(context, CL_MEM_READ_WRITE, filtroCPU.size() * sizeof(double));

cl::CommandQueue queue(context, default_device);

// COPIA DE DATOS A BUFFERS DE GPU
queue.enqueueWriteBuffer(sourceGPU, CL_TRUE, 0, sourceCPU.size() * sizeof(double), sourceCPU.data());
queue.enqueueWriteBuffer(filtroGPU, CL_TRUE, 0, filtroCPU.size() * sizeof(double), filtroCPU.data());
```

Ejecución del Kernel

```
// EJECUCION DEL KERNEL

cout << "Aplicando filtrado..." << endl;
cl::Kernel kernel_gauss = cl::Kernel(program, "applyFilter");
kernel_gauss.setArg(0, sourceGPU);
kernel_gauss.setArg(1, filtroGPU);
kernel_gauss.setArg(2, resultGPU);
kernel_gauss.setArg(3, height);
kernel_gauss.setArg(4, width);
kernel_gauss.setArg(5, filterHeight);
kernel_gauss.setArg(6, filterWidth);
queue.enqueueNDRangeKernel(kernel_gauss, cl::NullRange, cl::NDRange(height), cl::NullRange);
queue.finish();

queue.enqueueReadBuffer(resultGPU, CL_TRUE, 0, resultCPU.size() * sizeof(double), &resultCPU[0]);

cout << "Filtro aplicado" << endl;
                                                    int newImageWidth
Image resultMatrix(3, Matrix(newImageHeight, Array(newImageWidth)));

for (int d = 0; d < 3; d++)
{
    for (int i = 0; i < newImageHeight; i++)
    {
        for (int j = 0; j < newImageWidth; j++)
        {
            resultMatrix[d][i][j] = resultCPU[d * newImageHeight * newImageWidth + i * newImageWidth + j];
        }
    }
}
```

Posteriormente, se ha comprobado el resultado final obteniendo lo siguiente:

2. Analiza el rendimiento de las tres versiones paralelas que has desarrollado.

En primera instancia, podremos ver las imágenes utilizadas y la salida obtenida:

Imagen Pequeña 1200 x 630

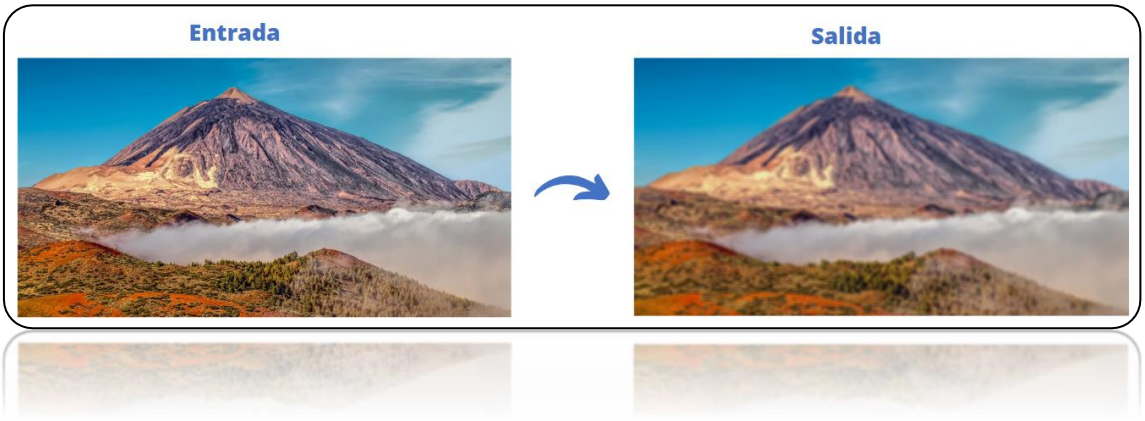


Imagen Mediana 2560 x 1440

Imagen Grande 4000 x 2250

A continuación, podremos ver los resultados obteniendo en la ejecución de los diferentes programas con diferente número de procesadores (si procede):

Secuencial

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/Secuencial$ ./a.out image.png
Loading image...
Applying filter...
Saving image...
Done!
Tiempo de ejecucion: 4.135 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/Secuencial$ ./a.out image.png
Loading image...
Applying filter...
Saving image...
Done!
Tiempo de ejecucion: 49.921 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/Secuencial$ ./a.out image.png
Loading image...
Applying filter...
Saving image...
Done!
Tiempo de ejecucion: 20.584 sec
```

MPI

Imagen Pequeña 1200 x 630

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 2 mpi_Final_version image.png 630 1200
Tiempo de ejecucion final: 0.798 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 3 mpi_Final_version image.png 630 1200
Tiempo de ejecucion final: 0.707 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 4 mpi_Final_version image.png 630 1200
Tiempo de ejecucion final: 0.691 sec
```

Imagen Mediana 2560 x 1440

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 2 mpi_Final_version image.png 1440 2560
Tiempo de ejecucion final: 3.586 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 3 mpi_Final_version image.png 1440 2560
Tiempo de ejecucion final: 3.276 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 4 mpi_Final_version image.png 1440 2560
Tiempo de ejecucion final: 3.19 sec
```

Imagen Grande 4000 x 2250

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 2 mpi_Final_version image.png 2250 4000
Tiempo de ejecucion final: 8.739 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 3 mpi_Final_version image.png 2250 4000
Tiempo de ejecucion final: 7.789 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/MPI$ make runFinal
mpiexec -np 4 mpi_Final_version image.png 2250 4000
Tiempo de ejecucion final: 7.601 sec
```

OPENMP

Imagen Pequeña 1200 x 630

```
ar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make ru
/OpenMP_version image.png 1

--- Información de la Imagen---
height: 630
width: 1200
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 4.279 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 2

--- Información de la Imagen---
height: 630
width: 1200
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 2.361 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 3

--- Información de la Imagen---
height: 630
width: 1200
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 1.714 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 4

--- Información de la Imagen---
height: 630
width: 1200
filterHeight: 10
filterWidth: 10

argando...
mpo de ejecucion: 1.367 sec
```

Imagen Mediana 2560 x 1440

```
r@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
/OpenMP_version image.png 1

--- Información de la Imagen---
height: 1440
width: 2560
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 21.17 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 2

--- Información de la Imagen---
height: 1440
width: 2560
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 12.191 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 3

--- Información de la Imagen---
height: 1440
width: 2560
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 8.475 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 4

--- Información de la Imagen---
height: 1440
width: 2560
filterHeight: 10
filterWidth: 10

argando...
empo de ejecucion: 7.184 sec
```

Imagen Grande 4000 x 2250

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make ru
./OpenMP_version image.png 1

--- Información de la Imagen---
height: 2250
width: 4000
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 51.082 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 2

--- Información de la Imagen---
height: 2250
width: 4000
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 28.305 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 3

--- Información de la Imagen---
height: 2250
width: 4000
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 20.656 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P3/OpenMP$ make run
./OpenMP_version image.png 4

--- Información de la Imagen---
height: 2250
width: 4000
filterHeight: 10
filterWidth: 10

Cargando...
Tiempo de ejecucion: 16.509 sec
```

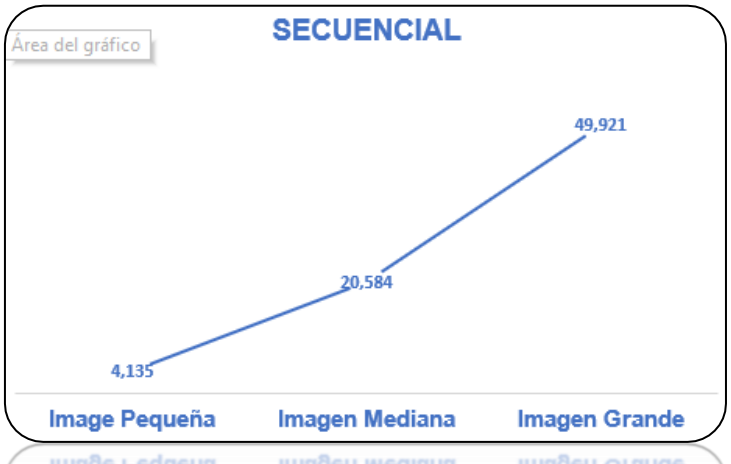
OPENCL

```
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P4/opencl$ make run
./OpenCL_version image.png
Cargando Imagen...
Using platform: Clover
Using device: Radeon RX 590 Series (POLARIS10, DRM 3.33.0, 5.3.0-45-generic, LLVM 9.0.0)
Aplicando filtrado...
Filtro aplicado
Guardando imagen...
Trabajo Finalizado
Tiempo de ejecucion: 2.982 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P4/opencl$ make run
./OpenCL_version image.png
Cargando Imagen...
Using platform: Clover
Using device: Radeon RX 590 Series (POLARIS10, DRM 3.33.0, 5.3.0-45-generic, LLVM 9.0.0)
Aplicando filtrado...
Filtro aplicado
Guardando imagen...
Trabajo Finalizado
Tiempo de ejecucion: 0.9 sec
omar@omar-B85M-D3H:~/Escritorio/ComputacionNube/P4/opencl$ make run
./OpenCL_version image.png
Cargando Imagen...
Using platform: Clover
Using device: Radeon RX 590 Series (POLARIS10, DRM 3.33.0, 5.3.0-45-generic, LLVM 9.0.0)
Aplicando filtrado...
Filtro aplicado
Guardando imagen...
Trabajo Finalizado
Tiempo de ejecucion: 5.895 sec
```

A continuación, podremos ver las gráficas generadas para cada solución propuesta

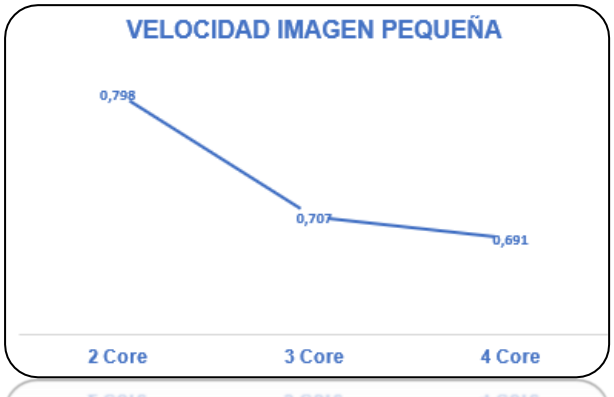
Secuencial

Gráfica Velocidad



MPI

Imagen Pequeña



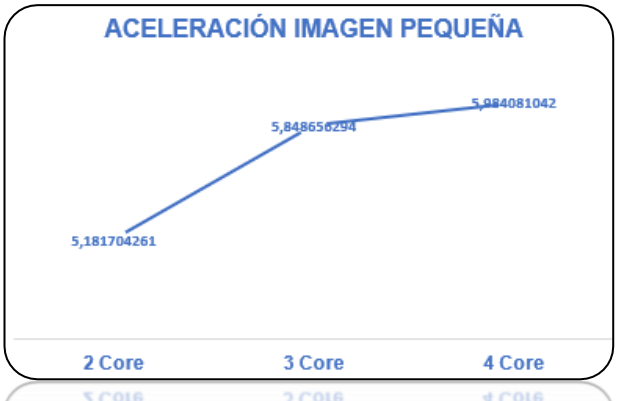
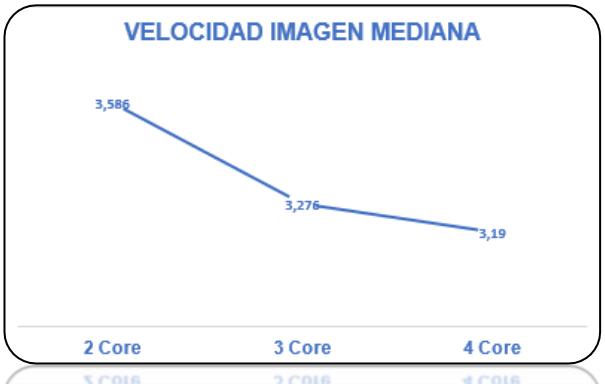


Imagen Mediana



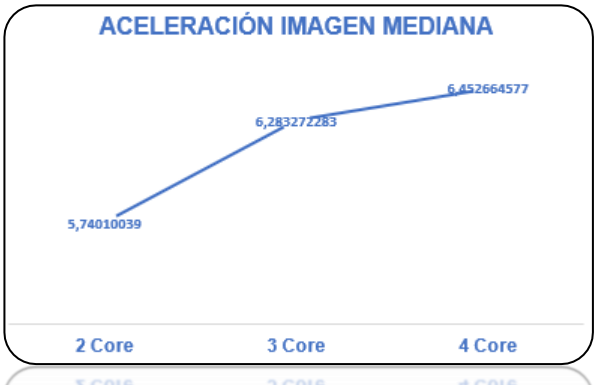
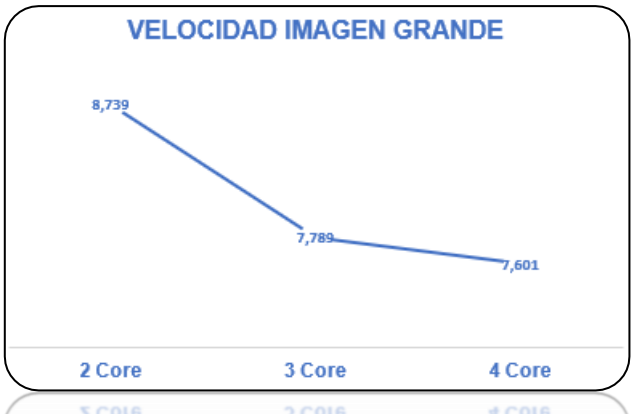
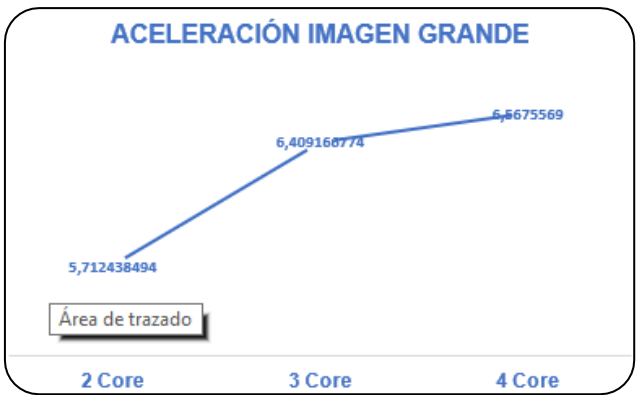
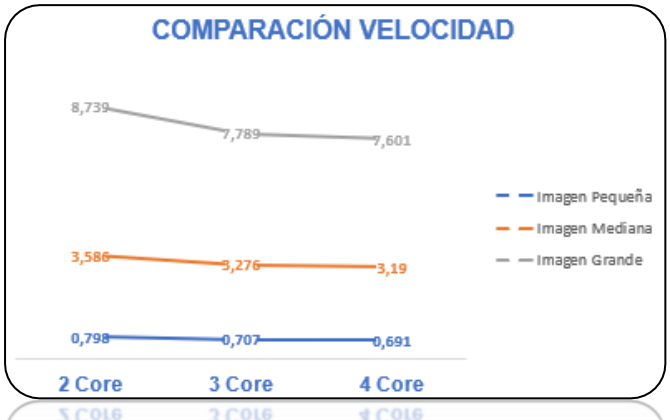


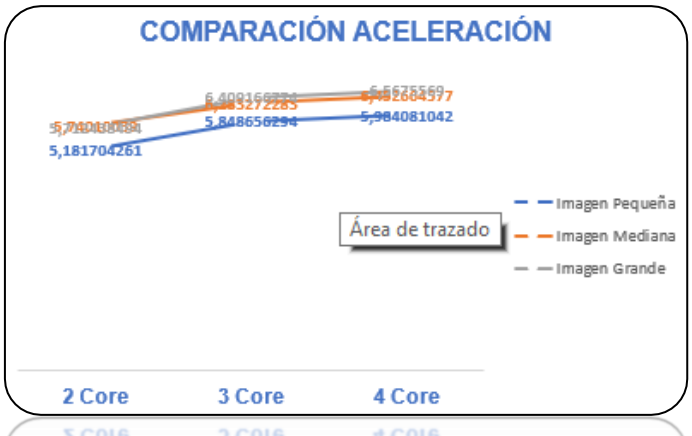
Imagen Grande





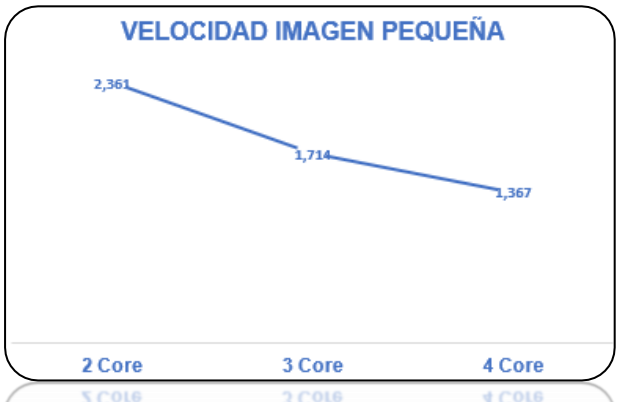
Resumen





OpenMP

Imagen Pequeña



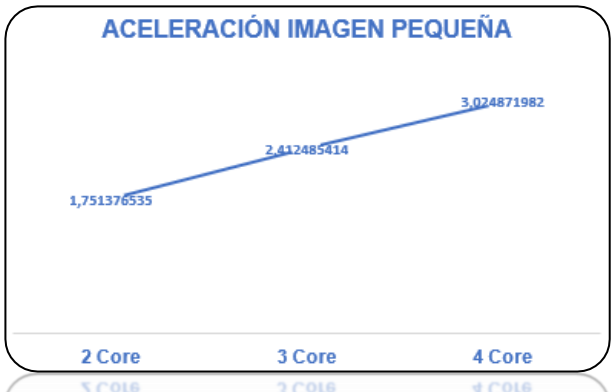
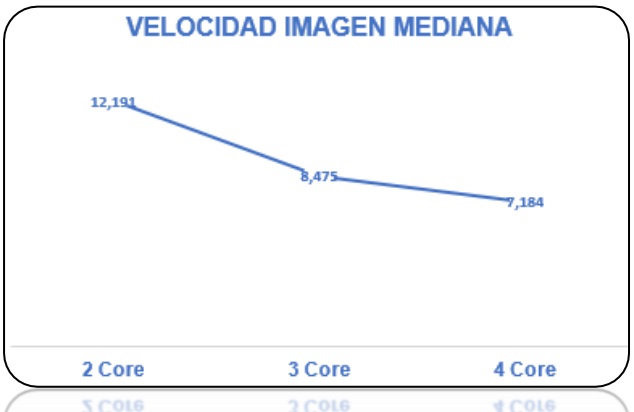


Imagen Mediana



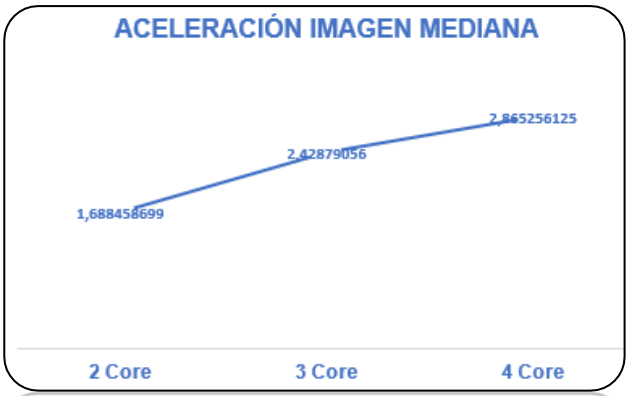
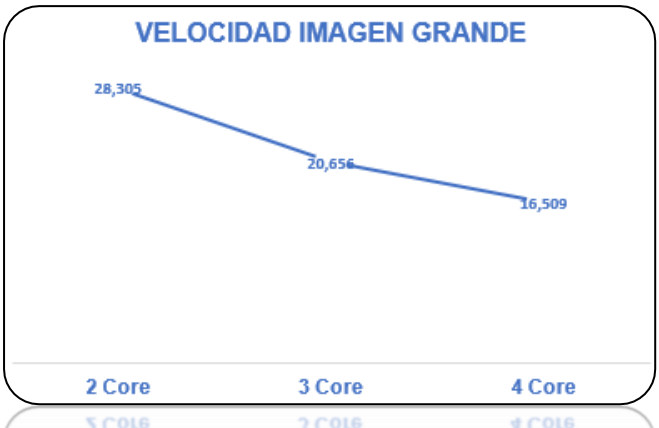
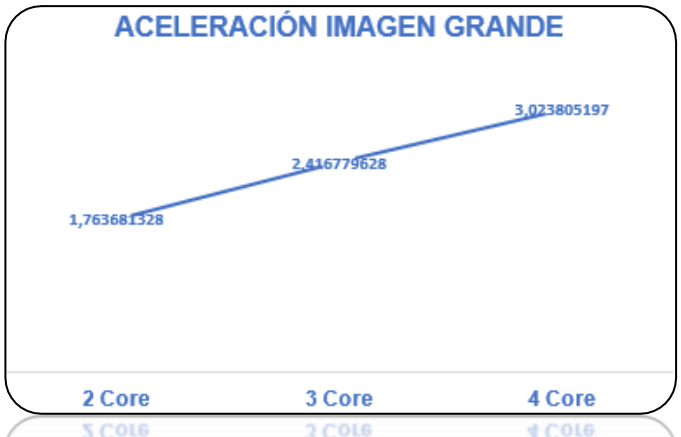
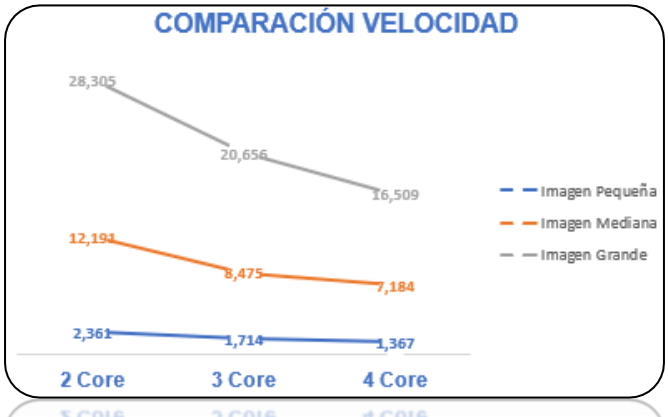


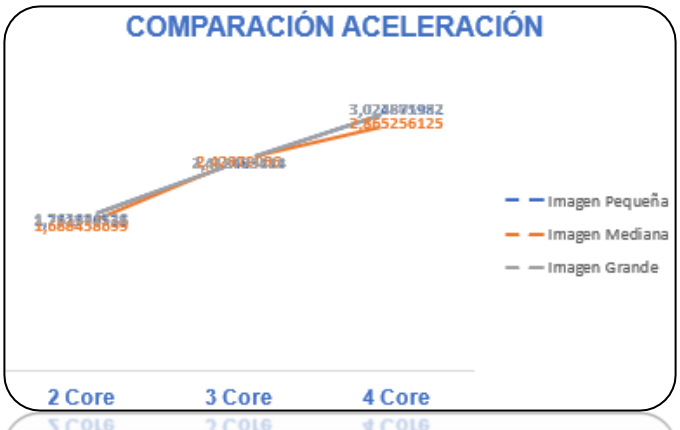
Imagen Grande





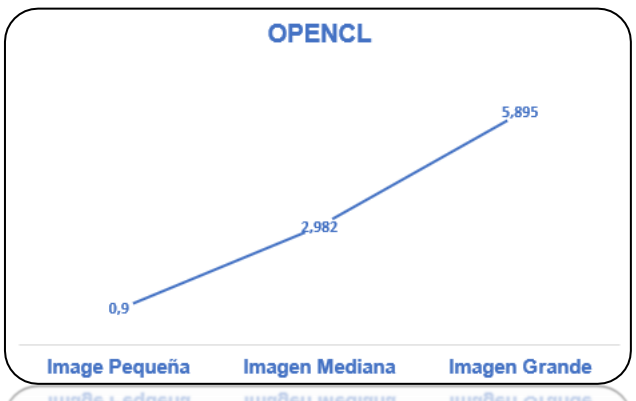
Resumen





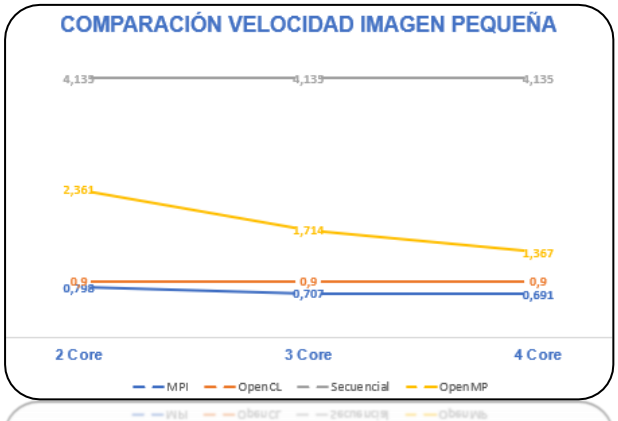
OpenCL

Gráfica Velocidad



RESUMEN GENERAL

Imagen Pequeña



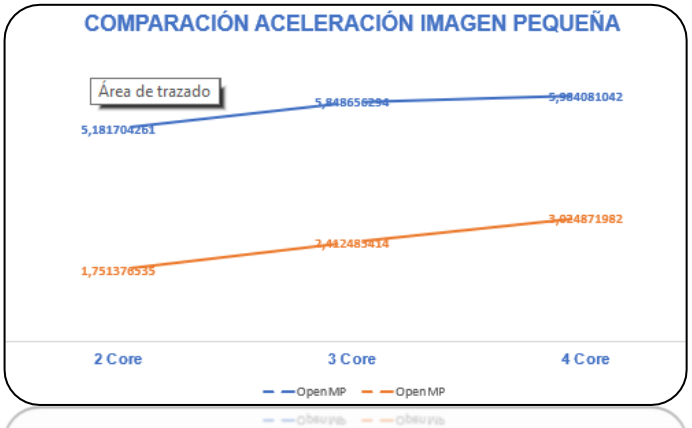
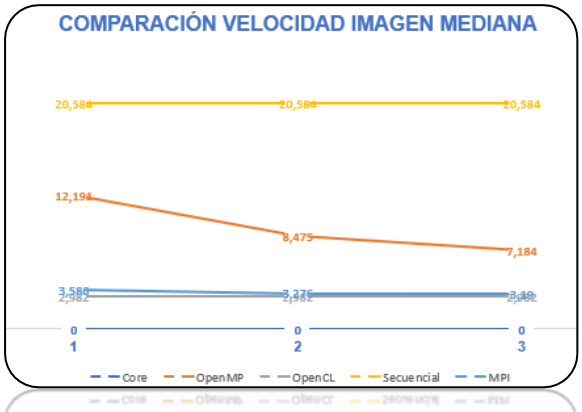


Imagen Mediana



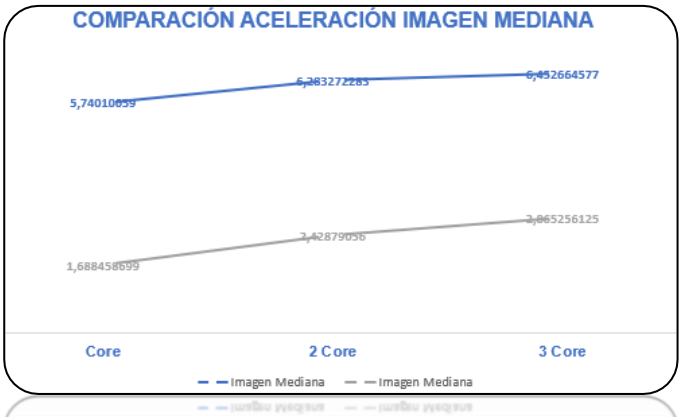
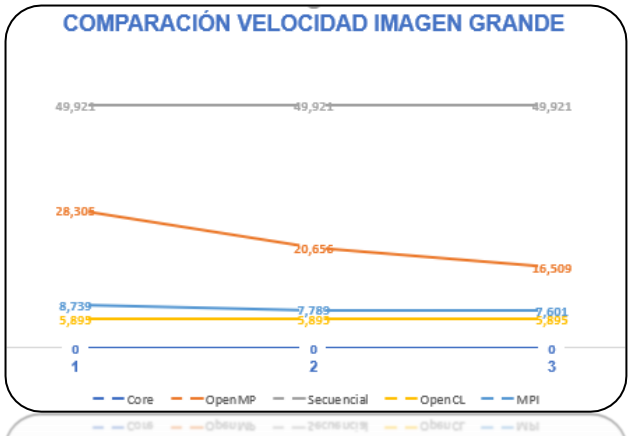
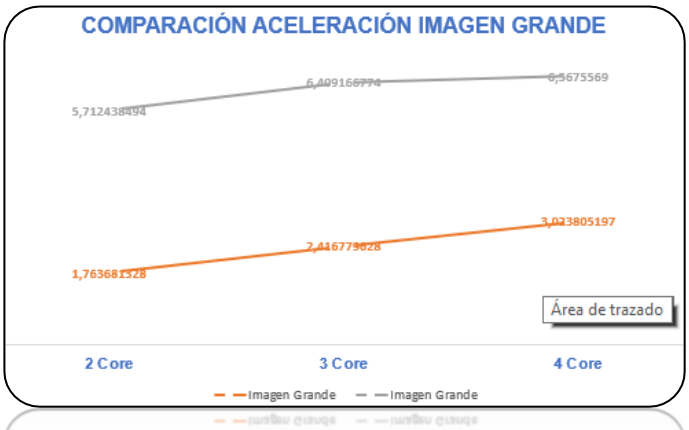


Imagen Grande





Las gráficas anteriores se han sacado con un procesador i7 que dispone de 4 cores. De igual forma, se ha usado una gráfica Radeon rx 590 Series. Por este motivo las gráficas llegan a un máximo de 4 cores

Como hemos visto en las gráficas anteriores, la velocidad de cómputo no es directamente proporcional al número de cores. Esto quiere decir que, si podemos ejecutar el código con un Core en 10 segundos, no vamos a poder reducir a la mitad dicho tiempo si ponemos un segundo Core. De igual forma, en la mayoría de los casos la pendiente de la aceleración es menor cada vez que subimos de Core, es decir, el programa se computa más rápido pero cada vez sube menos la aceleración.

Las gráficas anteriores nos permiten comparar las diferentes versiones de forma más sencilla. Gracias a esto podemos decir que, de las versiones creadas la más rápida es la de OpenCL seguida de MPI, OpenMP y secuencial (ordenadas por velocidad). Esto es lógico de forma teórica debido que los códigos paralelizados son más rápidos que los secuenciales. De igual forma, la computación con la gráfica es mucho más rápida para imágenes que el procesador. De la misma manera, es más rápido MPI que OpenMP. Esto es debido a que MPI es más configurable. Sin embargo, OpenMP es más fácil de programar. Por todo ello, opino que si se dispone de una buena gráfica es más rentable (pata este caso) usar OPENCL, Sin embargo, si no se puede usar tendríamos que seleccionar nuestra filosofía, es decir, si se requiere mayor velocidad o simplicidad de programación.